



Microcontroller Systems

ELET 3232

Topic 19: Timer/Counter with PWM



Objectives

- To understand the operation of an 8 bit timer/counter in an AVR microcontroller
- To understand how to use the timer/counter to generate a PWM signal
- To understand how PWM signals are used to control servo motors
 - Position control
 - Speed control



Modes of Operation

- Four modes of operation for Timer/Counter 0
 - Normal Mode:
 - we have used this mode to develop timing loops
 - Phase Correct PWM:
 - The mode we will use in PWM
 - CTC Mode:
 - Clear Timer on Compare Match
 - Fast PWM:
 - Fast Pulse Width Modulation



[Modes of Operation]

- We will discuss:
 - Normal Mode on Timer/Counter 0 (8 bit counter):
 - we have used this mode to develop timing loops
 - Phase Correct PWM on Timer/Counter 0:
 - The mode we will use in PWM
 - Phase Correct PWM on Timer/Counter 1:
 - A 16-bit Timer/Counter



Normal Mode

■ Example Timing Loop

```
Init:      ldi    r16,0x07      ; set the prescaler for the counter
           out    TCCR0,r16     ; prescaler = 1024 (ATmega128)
;=====
TDelay:
           ldi    r16,0         ; initialize timer/counter 0
           out    TCNT0,r16     ; starts the counter

TLoop:     in     r17,TCNT0      ; read timer/counter value
           cpi    r17,0x63      ; is it $63?
           brne   TLoop         ; if not $63, read it again (until $63)

           done
;=====
```



Normal Mode

TCCR0

```

Init:      ldi    r16,0x07      ; set the prescaler for the counter
           out    TCCR0,r16     ; prescaler = 1024 (ATmega128)
;=====
TDelay:
           ldi    r16,0         ; initialize timer/counter 0
           out    TCNT0,r16     ; starts the counter

TLoop:     in     r17,TCNT0      ; read timer/counter value
           cpi    r17,0x63      ; is it $63?
           brne   TLoop         ; if not $63, read it again (until $63)

           done
;=====
  
```

Step 1: Initialize the Timer/Counter

We are most concerned with bits 2, 1, and 0. They control the prescaler

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Normal Mode

TCCR0

Table 56. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{T0S}}/(\text{No prescaling})$
0	1	0	$\text{clk}_{\text{T0S}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{T0S}}/32$ (From prescaler)
1	0	0	$\text{clk}_{\text{T0S}}/64$ (From prescaler)
1	0	1	$\text{clk}_{\text{T0S}}/128$ (From prescaler)
1	1	0	$\text{clk}_{\text{T0S}}/256$ (From prescaler)
1	1	1	$\text{clk}_{\text{T0S}}/1024$ (From prescaler)

We can turn off the counter

We can use the system clock or

We can use a slower signal based on the system clock

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Normal Mode

TCCR0

Figure 42. Timer/Counter Timing Diagram



System Clock (assume 10MHz)

If we used the system clock (no prescaling) to increment the counter, it would be incremented every 100ns

$$T = 1/f_{\text{clk}} = 1/10\text{MHz} = 100\text{ns}$$

Timer/Counter Control Register – TCCR0

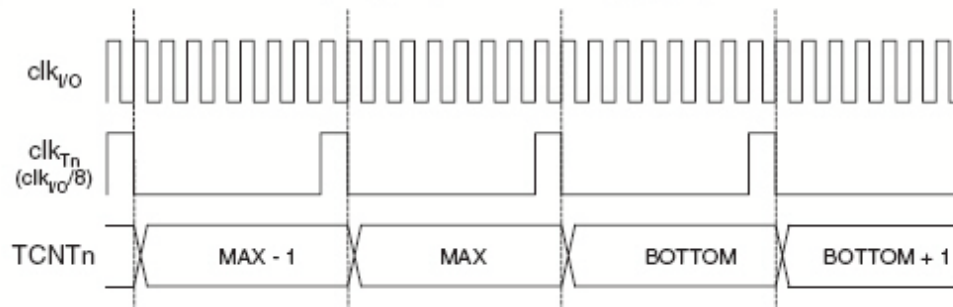
Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Normal Mode

TCCR0

Figure 42. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)



System Clock (assume 10MHz)

If we used a prescaler of 8 to increment the counter, it would be incremented every 8th clock pulse

$$T_{clk} * 8 = 100ns * 8 = 800ns$$

Timer/Counter Control Register – TCCR0

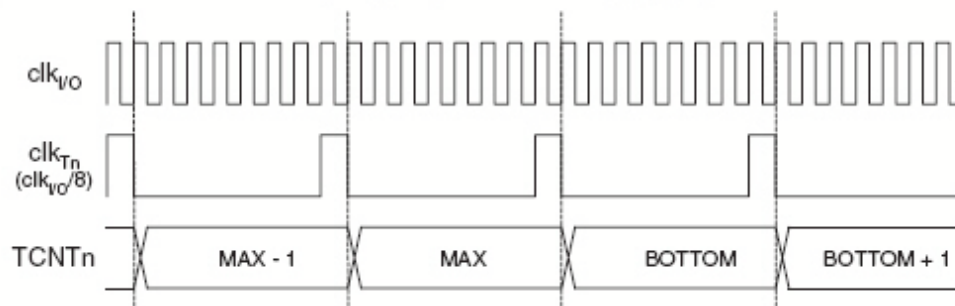
Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Normal Mode

TCCR0

Figure 42. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)



System Clock (assume 10MHz)

If we used a prescaler of 1024 to increment the counter, it would be incremented every 1024th clock pulse

$$T_{clk} * 1024 = 100ns * 1024 = 102.4us$$

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Normal Mode

Example Timing Loop

```

Init:      ldi    r16,0x07      ; set the prescaler for the counter
           out    TCCR0,r16     ; prescaler = 1024 (ATmega128)
;=====
TDelay:
           ldi    r16,0         ; initialize timer/counter 0
           out    TCNT0,r16     ; starts the counter

TLoop:     in     r17,TCNT0      ; read timer/counter value
           cpi    r17,0x63      ; is it $63?
           brne   TLoop         ; if not $63, read it again (until $63)

           done
;=====
  
```

Step 1: Initialize the Timer/Counter
 We used a prescaler of 1024, so bits 0-7 in the Timer/Counter Control Register were set to 1

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Normal Mode

■ Example Timing Loop

```
Init:      ldi    r16,0x07      ; set the prescaler for the counter
           out    TCCR0,r16     ; prescaler = 1024 (ATmega128)
;=====
TDelay:
           ldi    r16,0         ; initialize timer/counter 0
           out    TCNT0,r16     ; starts the counter

TLoop:     in     r17,TCNT0      ; read timer/counter value
           cpi    r17,0x63      ; is it $63?
           brne   TLoop         ; if not $63, read it again (until $63)

           done
;=====
```

Step 1: Initialize the Timer/Counter

Step 2: Start the Timer/Counter: placing a 0 in the Timer/Counter Register starts it



Normal Mode

Example Timing Loop

```
Init:      ldi    r16,0x07      ; set the prescaler for the counter
           out    TCCR0,r16     ; prescaler = 1024 (ATmega128)
;=====
TDelay:
           ldi    r16,0         ; initialize timer/counter 0
           out    TCNT0,r16     ; starts the counter

TLoop:     in     r17,TCNT0      ; read timer/counter value
           cpi    r17,0x63      ; is it $63?
           brne   TLoop         ; if not $63, read it again (until $63)

           done
;=====
```

Step 1: Initialize the Timer/Counter

Step 2: Start the Timer/Counter

Step 3: check for the calculated value

In this case, the Timer/Counter will increment its count every 1024 clock pulses (every 102.4 us). This loop keeps checking until the Timer/Counter gets to 99 (0x63). So the timer delays for:

$$100 * 102.4 \text{ us} = 10.24\text{ms}$$

Note: count from 0 to 99 is 100



[Phase Correct PWM]

- We will use Phase Correct PWM most of the time
 - PWM for short (as opposed to fast or CTC modes)



[Phase Correct PWM¹

- We will need the following registers:
 - TCCR0: Timer/Counter Control Register 0
 - TNCT0: Timer/Counter 0
 - OCR0: Output Compare Register 0
- The generated PWM waveform will appear on the OC0 pin

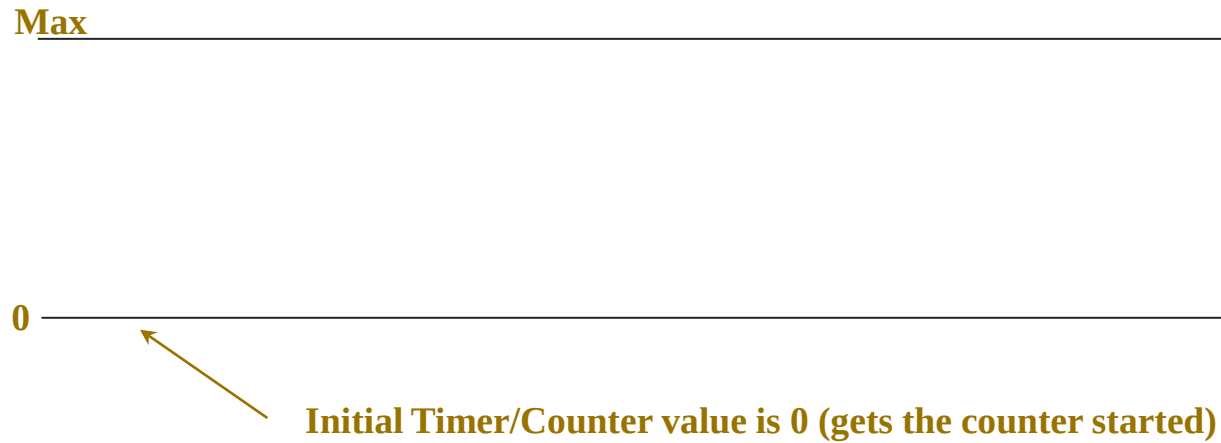


[Phase Correct PWM]

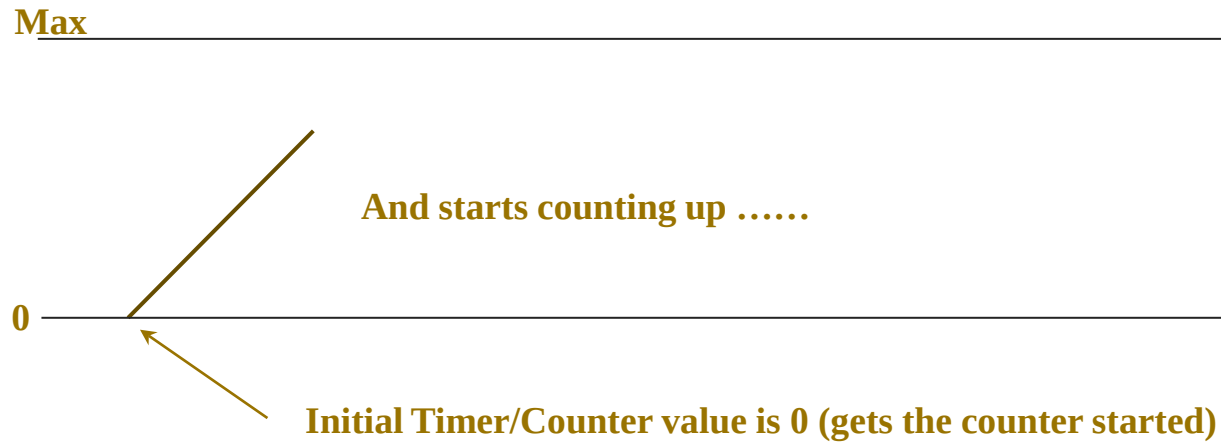
- We will need the following registers:
 - TCCR0: Timer/Counter Control Register 0
 - To initialize the Timer/Counter and set the prescaler
 - TNCT0: Timer/Counter 0
 - To hold the value of the counter
 - OCR0: Output Compare Register 0
 - Holds the count value for which the PWM signal will toggle



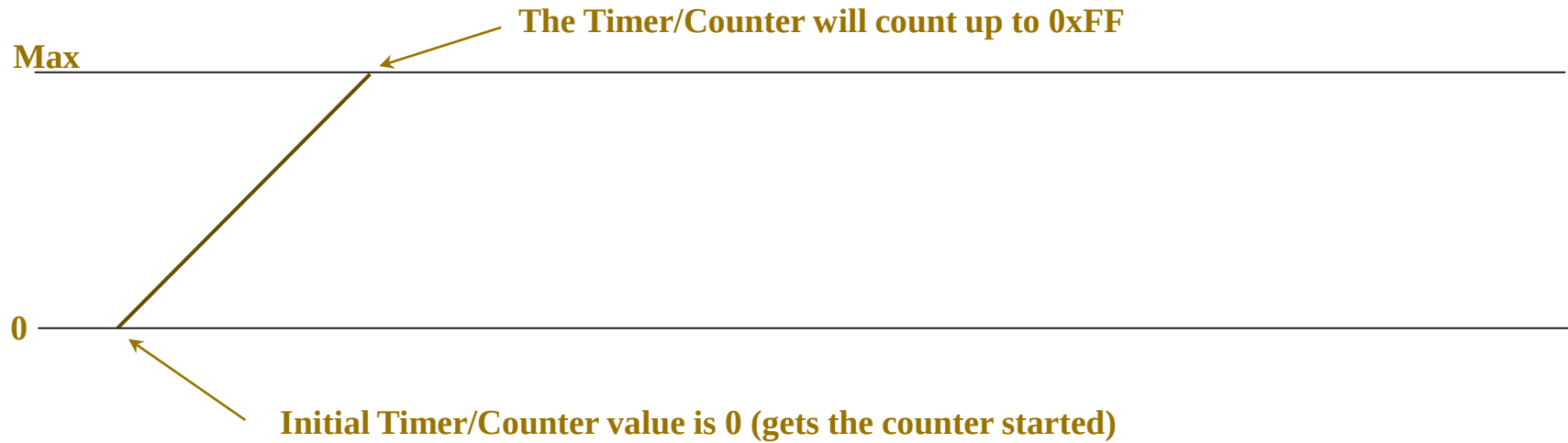
[Phase Correct PWM]



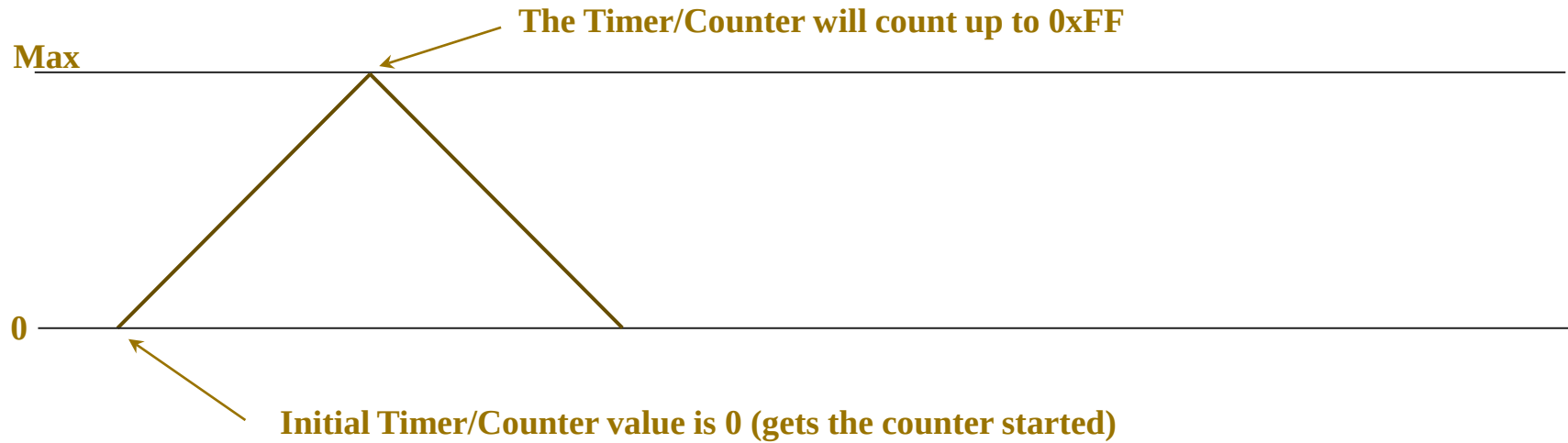
[Phase Correct PWM]



[Phase Correct PWM]



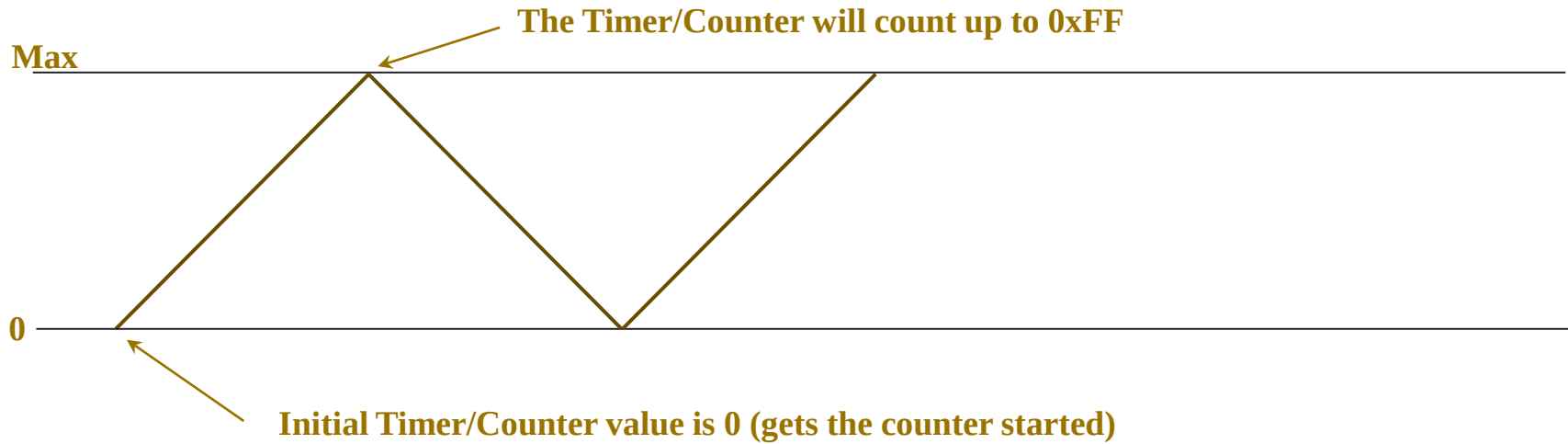
[Phase Correct PWM]



Then it will count down to 0



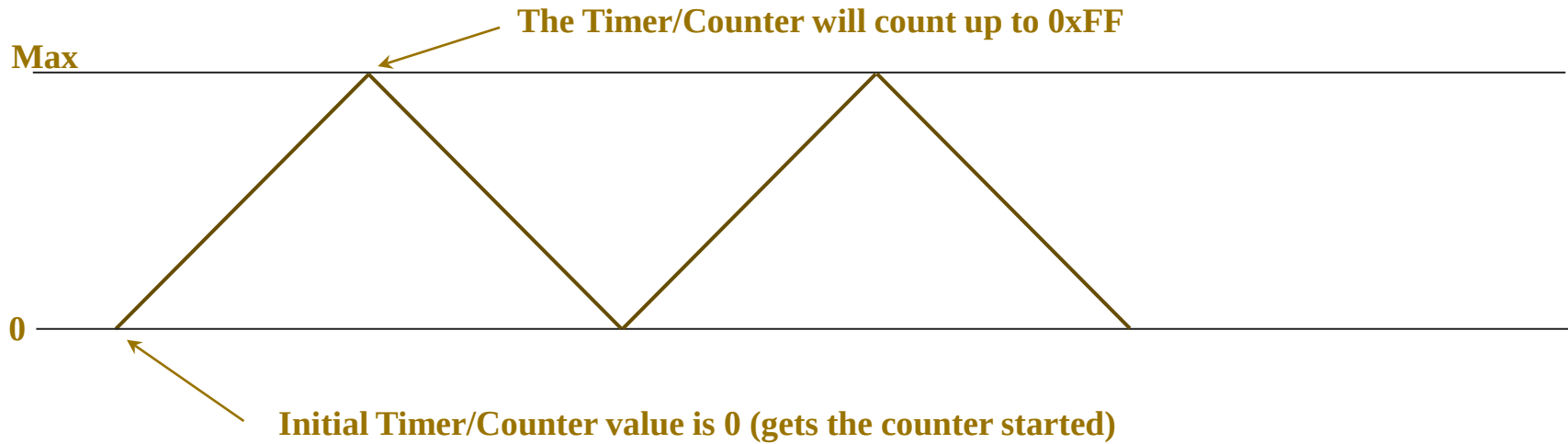
[Phase Correct PWM]



Then it will count down to 0, then up to some maximum



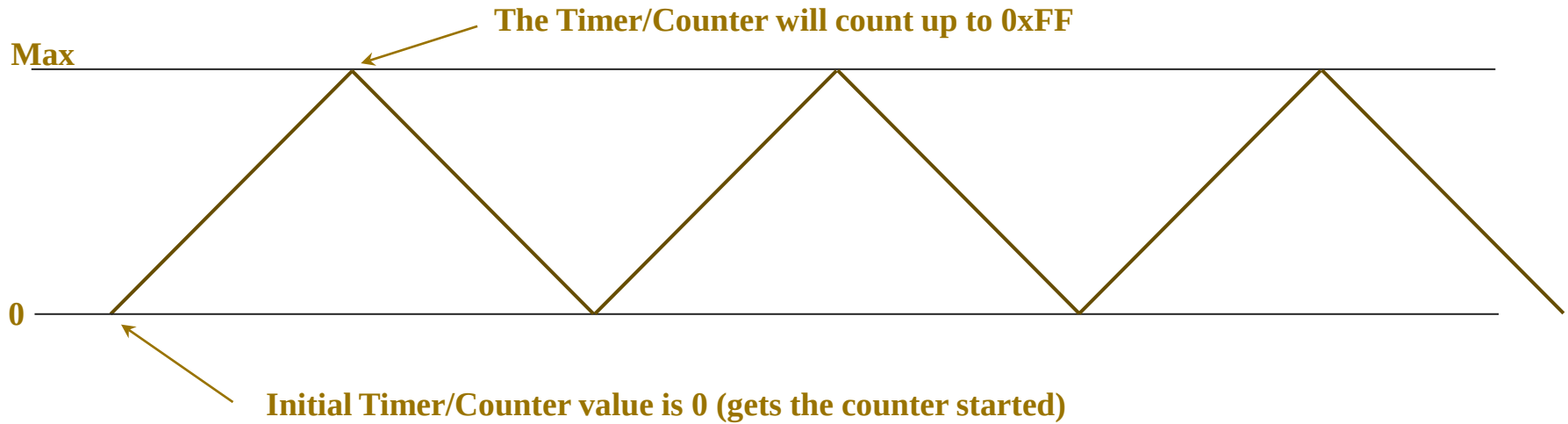
[Phase Correct PWM]



Then it will count down to 0, then up to some maximum, then down to 0



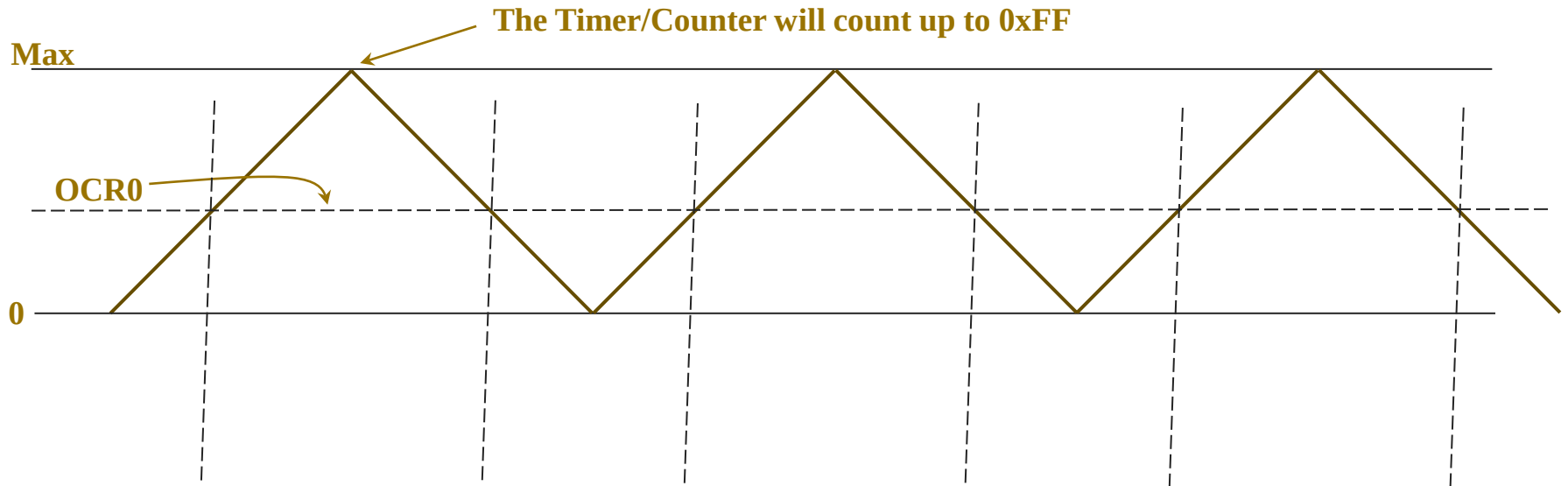
[Phase Correct PWM]



Then it will count down to 0, then up to some maximum, then down to 0, etc.



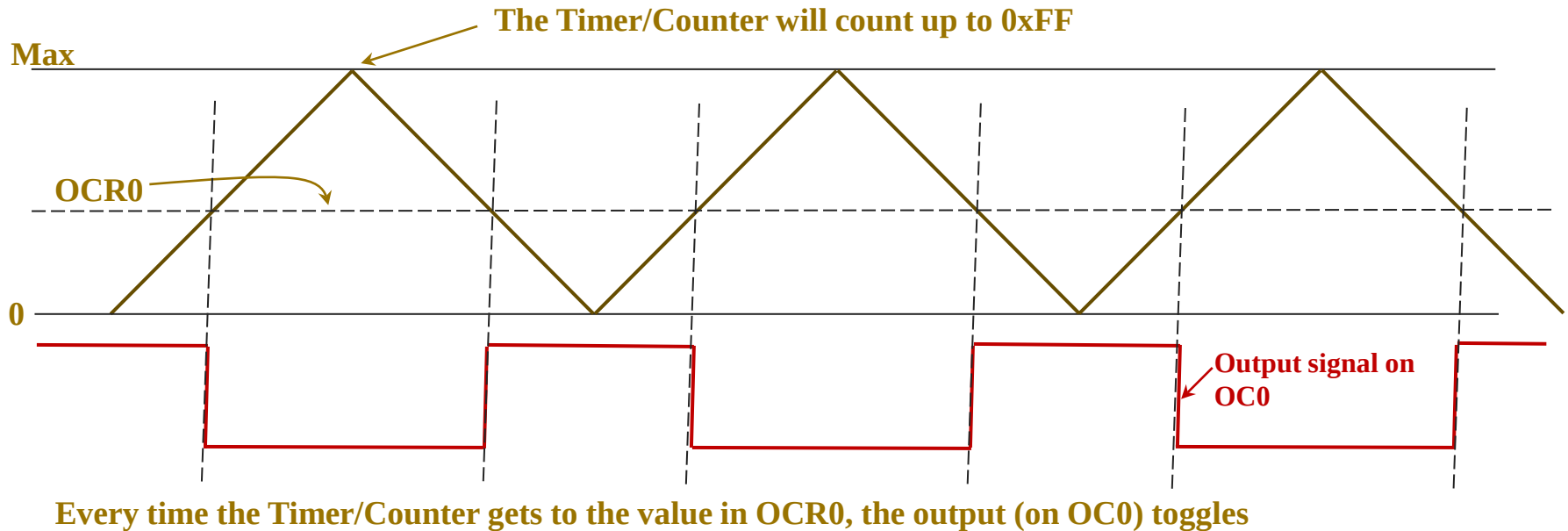
[Phase Correct PWM]



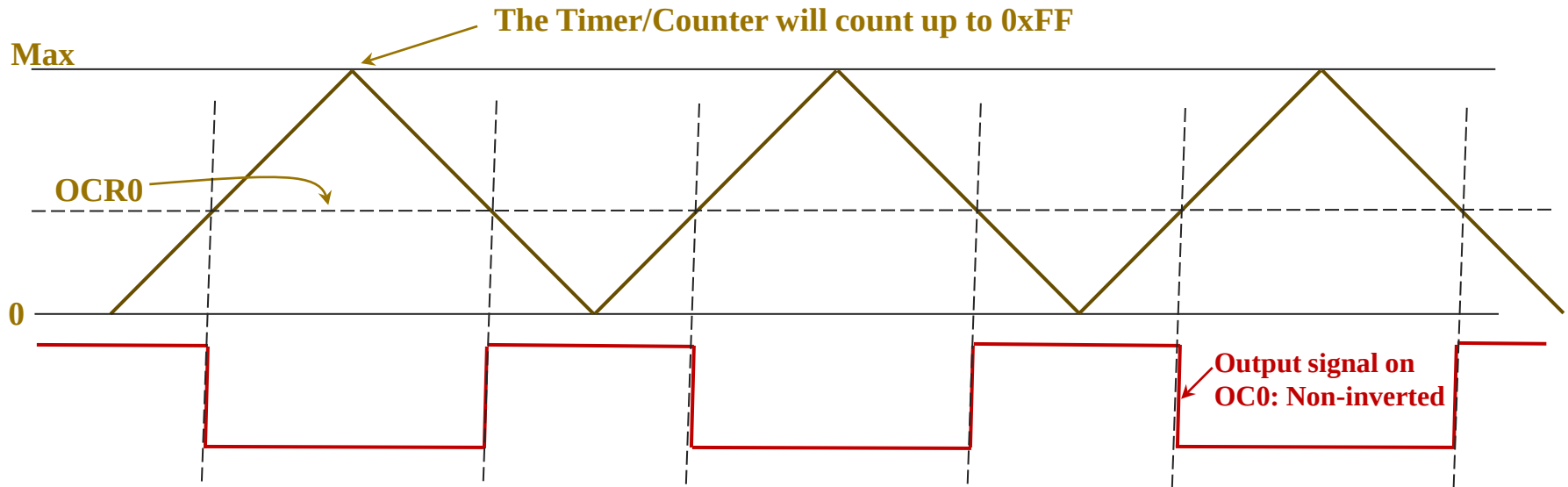
The timer will count up, eventually getting to the value in OCR0. It will then continue to count to 0xFF and then start counting down, eventually getting to the value in OCR0 again, then continuing its count down to 0.



[Phase Correct PWM]



[Phase Correct PWM]



Every time the Timer/Counter gets to the value in OCR0, the output (on OC0) toggles

Setting the COM01:0 bits (in TCCR0) to 2 will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM01:0 to 3

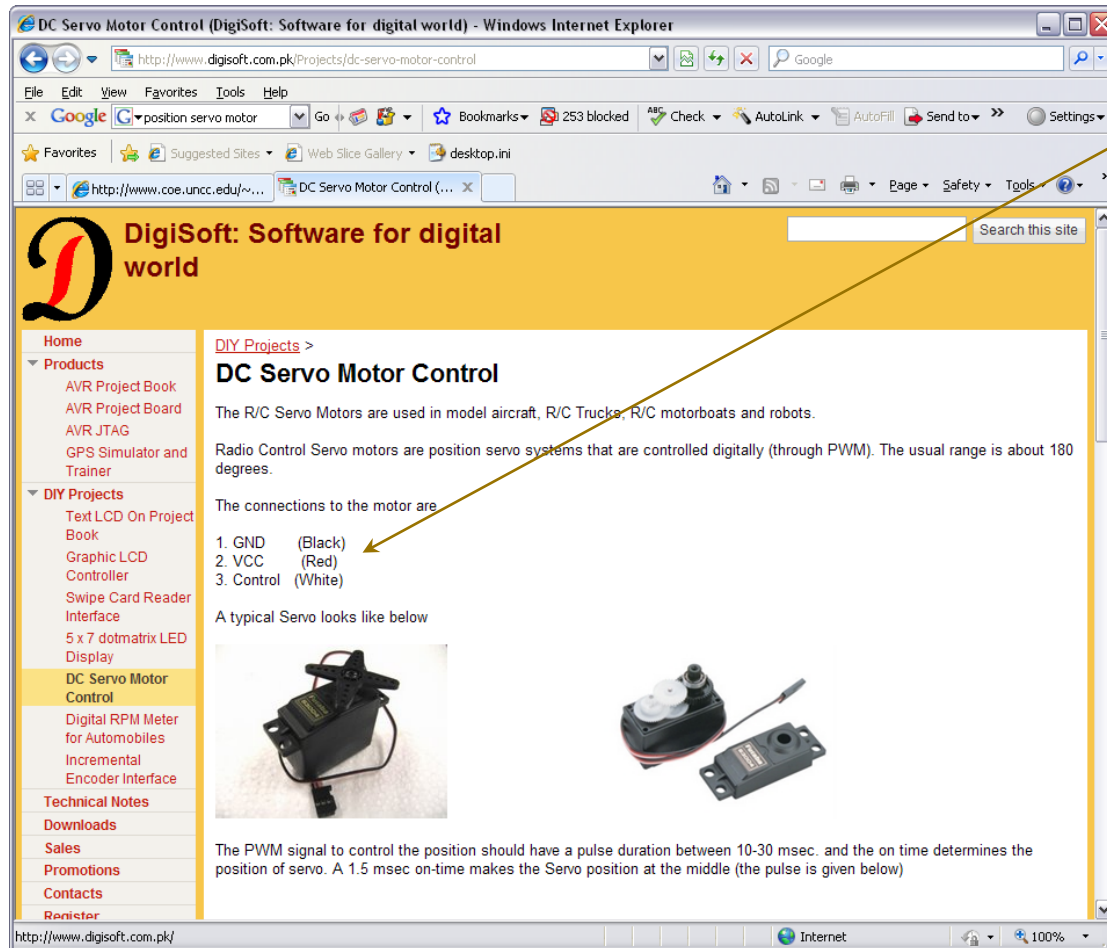


Calculations

- The big questions:
 - How do we calculate the values:
 - How do we calculate which prescaler value to use?
 - How do we calculate the value OCR0?
- Let's assume we are using the positioning servo motor shown on the next page²



Calculations



3 pins: Gnd, Vcc, and Control

The PWM signal should have a period of 10 to 30ms.



Calculations

DC Servo Motor Control (DigiSoft: Software for digital world) - Windows Internet Explorer

http://www.digisoft.com.pk/Projects/dc-servo-motor-control

File Edit View Favorites Tools Help

Google position servo motor Go Bookmarks 253 blocked Check AutoLink AutoFill Send to Settings

Favorites Suggested Sites Web Slice Gallery desktop.ini

http://www.coe.uncc.edu/~... DC Servo Motor Control (...)

DigiSoft: Software for digital world

Home

Products

- AVR Project Book
- AVR Project Board
- AVR JTAG
- GPS Simulator and Trainer

DIY Projects

- Text LCD On Project Book
- Graphic LCD Controller
- Swipe Card Reader Interface
- 5 x 7 dotmatrix LED Display
- DC Servo Motor Control**
- Digital RPM Meter for Automobiles
- Incremental Encoder Interface

Technical Notes

Downloads

Sales

Promotions

Contacts

Register

DIY Projects >

DC Servo Motor Control

The R/C Servo Motors are used in model aircraft, R/C Trucks, R/C motorboats and robots.

Radio Control Servo motors are position servo systems that are controlled digitally (through PWM). The usual range is about 180 degrees.

The connections to the motor are

1. GND (Black)
2. VCC (Red)
3. Control (White)

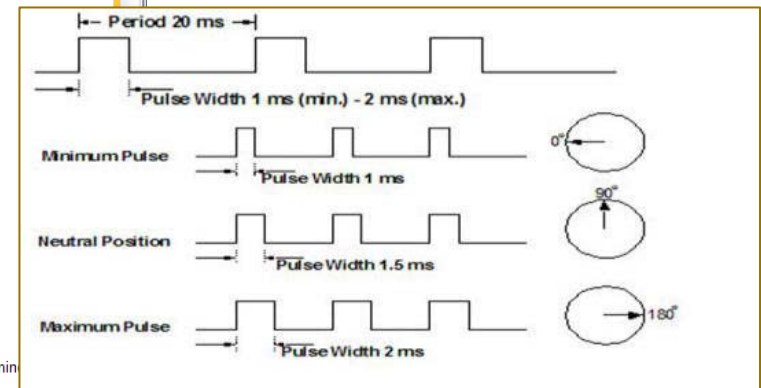
A typical Servo looks like below



The PWM signal to control the position should have a pulse duration between 10-30 msec. and the on time determines position of servo. A 1.5 msec on-time makes the Servo position at the middle (the pulse is given below)

3 pins: Gnd, Vcc, and Control

The PWM signal should have a period of 10 to 30ms. A pulse of 1.5ms sets the neutral position (90°), 1ms sets the left position (0°) and 2ms sets the right position (180°)



Calculations

- To calculate the prescaler, the ATMega data sheet gives the following formula (pg102)

$$f_{oc0} = \frac{f_{clk}}{N * 510}$$



Calculations

- To calculate the prescaler, the ATMega data sheet gives the following formula (pg102)

$$f_{oc0} = \frac{f_{clk}}{N * 510}$$

Assuming we have a system clock of 10MHz and we are using a prescaler of 1024, we would have:

$$f_{oc0} = \frac{10 \times 10^6}{1024 * 510}$$

$$f_{oc0} = 19.15Hz \quad T_{oc0} = 52.2ms$$

Too high



Calculations

- To calculate the prescaler, the ATMega data sheet gives the following formula (pg102)

$$f_{oc0} = \frac{f_{clk}}{N * 510}$$

Assuming we have a system clock of 10MHz and we are using a prescaler of 1024, we would have:

$$f_{oc0} = \frac{10 \times 10^6}{256 * 510}$$

$$f_{oc0} = 76.59Hz \quad T_{oc0} = 13.05ms$$



Calculations

- If we had a system clock of 16MHZ, we would have:

$$f_{oc0} = \frac{16 \times 10^6}{1024 * 510}$$

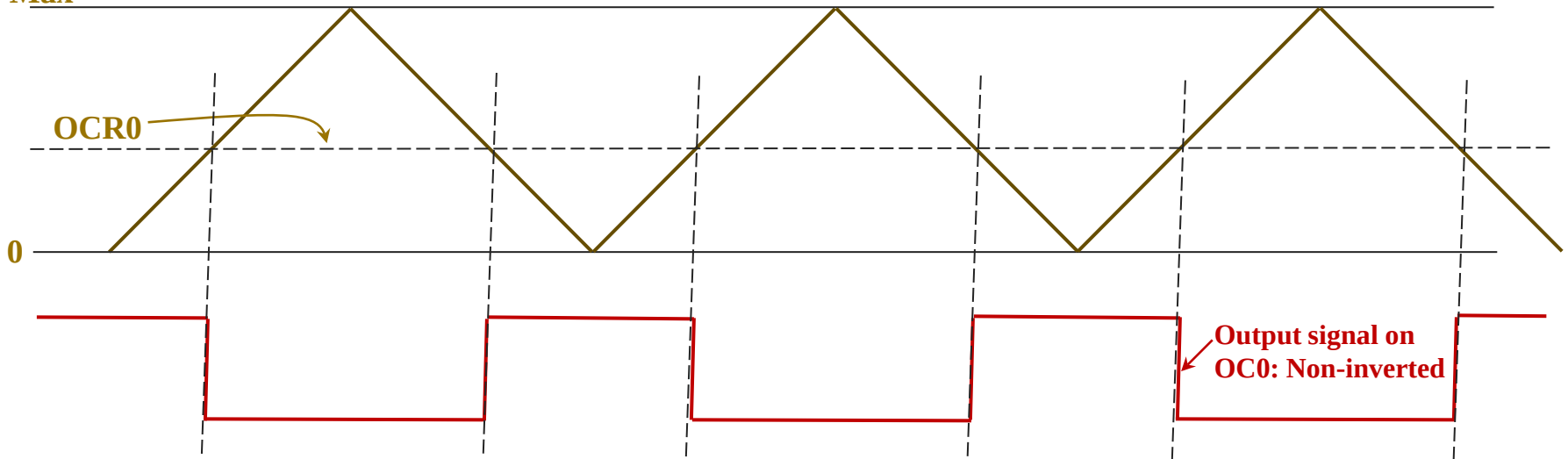
$$f_{oc0} = 30.63Hz \quad T_{oc0} = 32.64ms$$

A little too long. But, let's go back to the 10MHz system clock and the 77Hz (13ms period) OC0 clock



[Phase Correct PWM]

Max



It's probably easier to use a non-inverted signal.

Set COM01:0 to 2 (in TCCR0), CS02:0 to 7 for a prescaler of 1024, and WGM01 to 0 and WGM00 to 1 for PC PWM.

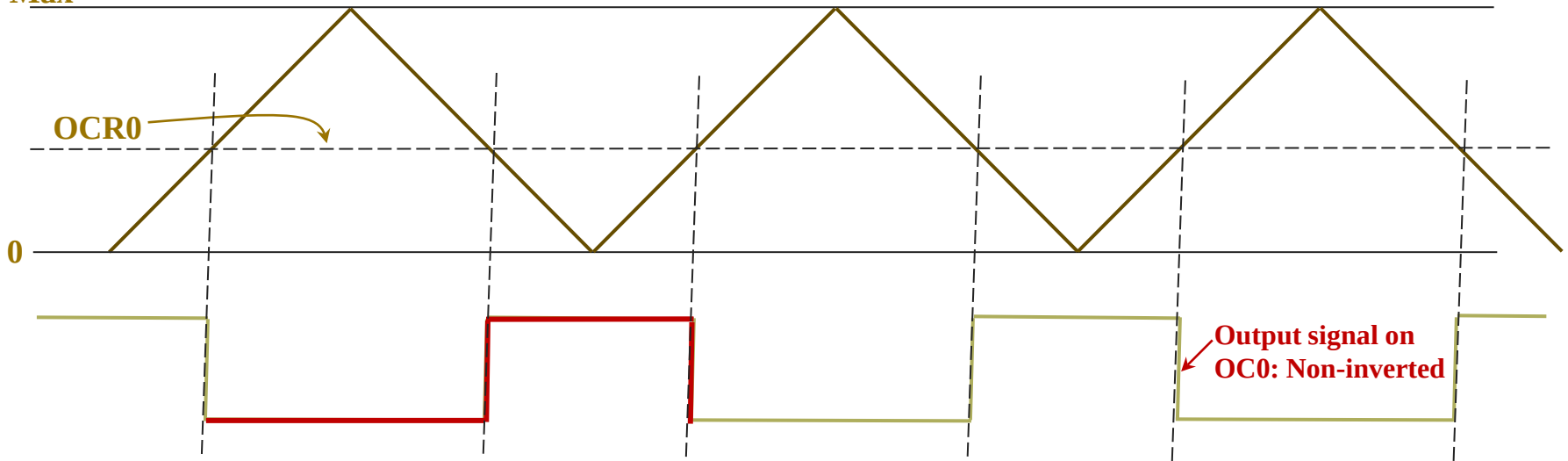
Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



[Phase Correct PWM]

Max



A full count: from OCR0 to Max, down to 0 (through OCR0), and then back up to OCR0 would be one cycle (highlighted above).

This represents a count of 256 twice – or a count to 512 – which represents 13ms (1/77Hz), so

- a count to ≈ 40 represents about 1 ms (0°) count from 20 down to 0 and up to 20 (total of 40) **OCR0=20**
- a count to ≈ 59 represents about 1.5ms (90°) count from 30 down to 0 and then up to 30 (total of 60) **OCR0=30**
- a count to ≈ 78 represents about 2ms (180°) count from 39 down to 0 and up to 39 (total of 78) **OCR0=39**



[

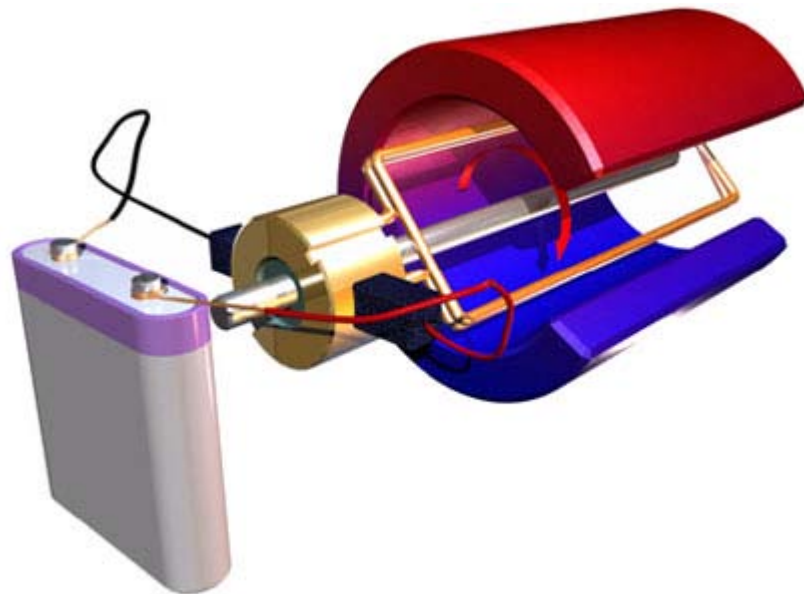
]

Speed Control



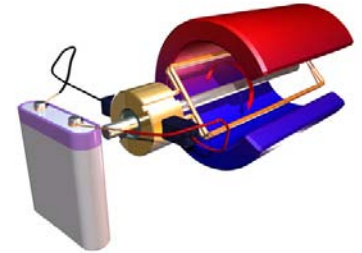
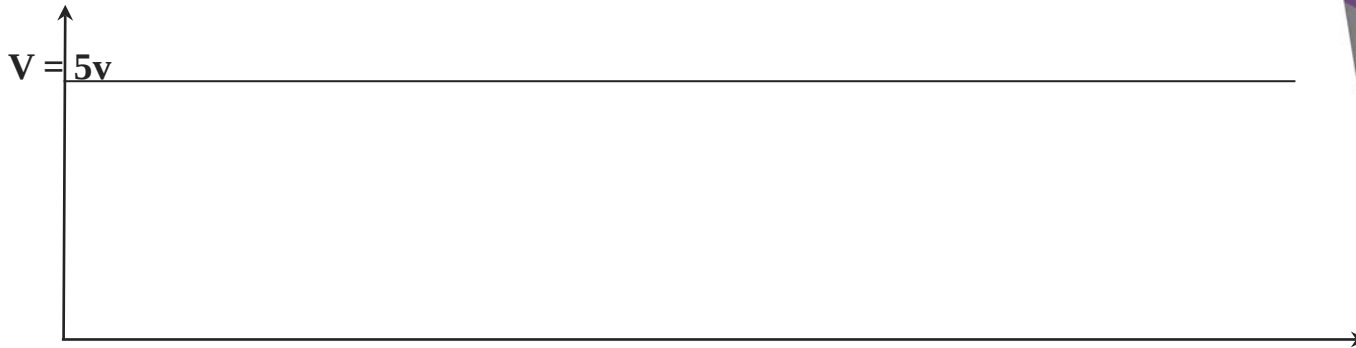
[PWM for Speed Control]

- A DC motor needs a voltage (and current) to make it rotate

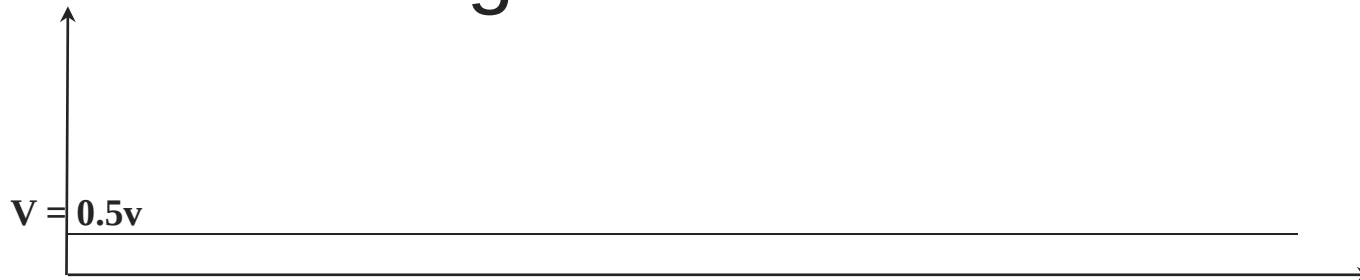


[PWM for Speed Control]

- A high voltage: it rotates faster



- Lower voltage: it rotates slower



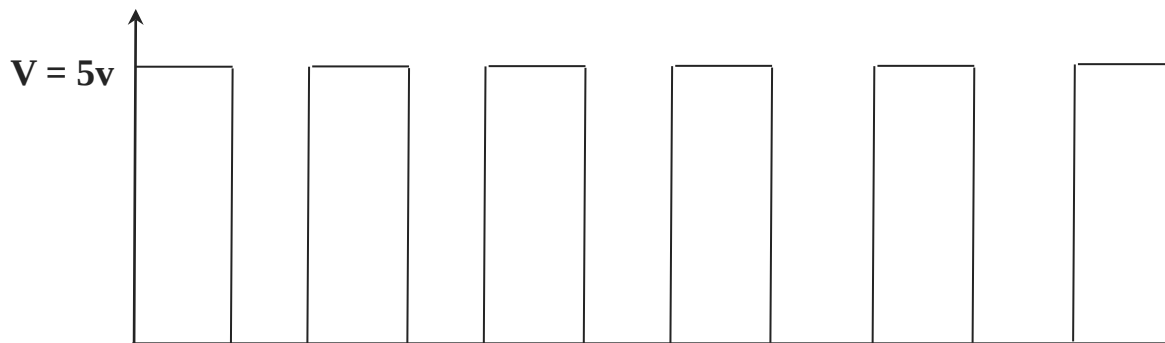
[PWM for Speed Control]

- But the AVR will output 5v or 0, not 1.5 or 3.2, etc
- Are we limited to “full speed” or “stop”



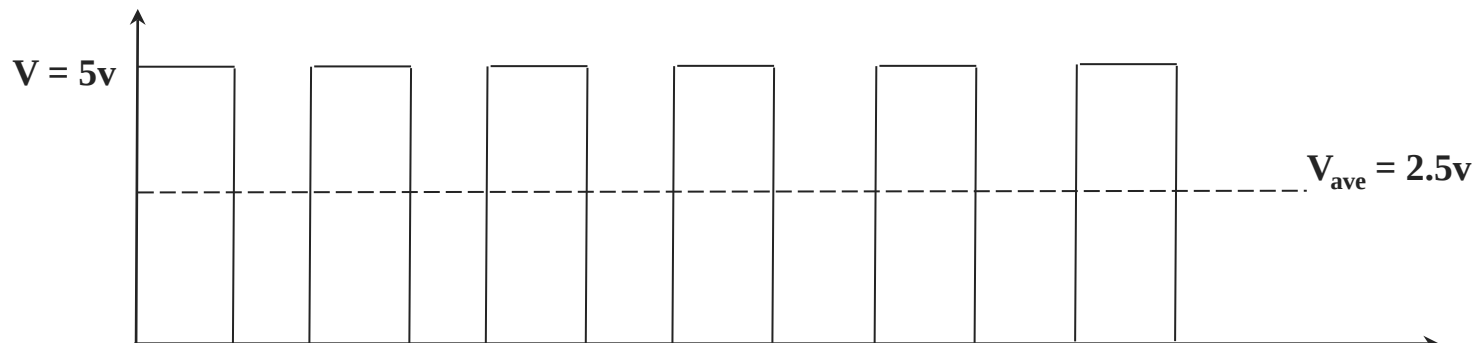
[PWM for Speed Control]

- But the AVR will output 5v or 0, not 1.5 or 3.2, etc
- What is the (approximate) average voltage of the waveform below (assume 50% duty cycle):



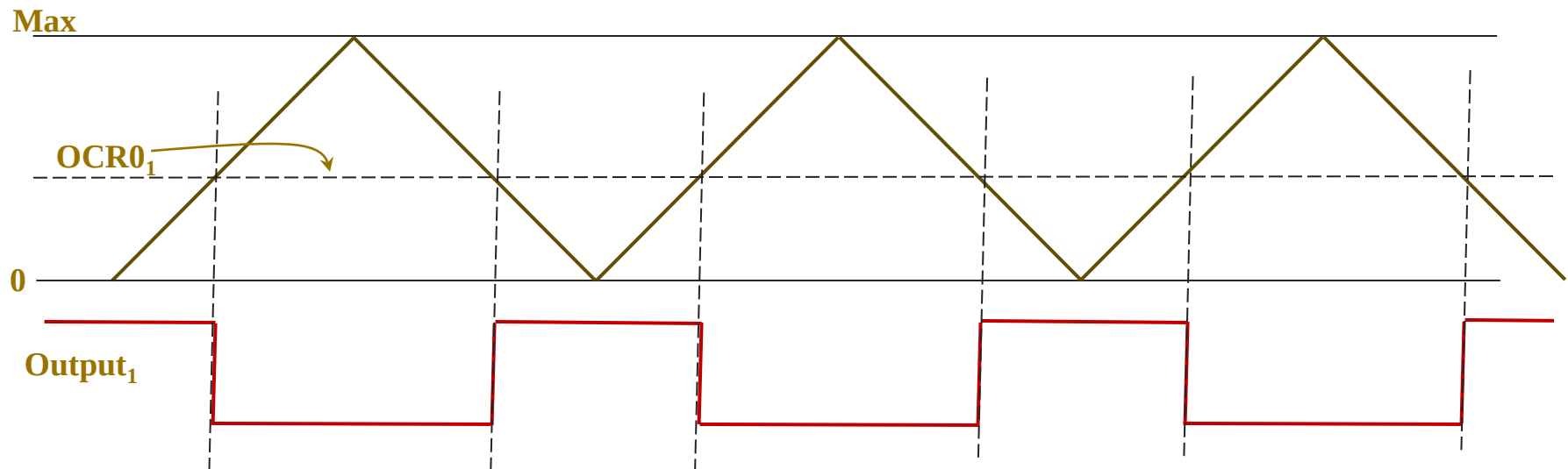
[PWM for Speed Control]

- But the AVR will output 5v or 0, not 1.5 or 3.2, etc
- What is the (approximate) average voltage of the waveform below (assume 50% duty cycle):



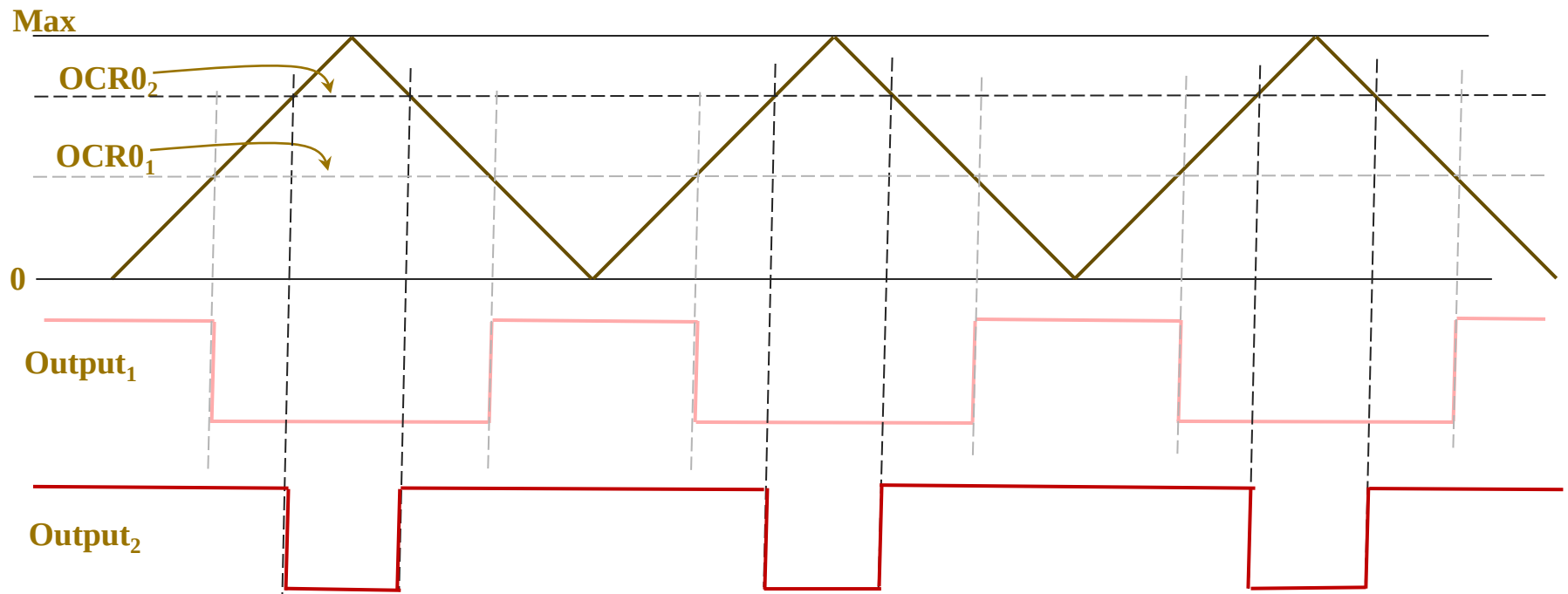
PWM for Speed Control

- We can change the average voltage of the PWM signal by changing the value in OCR0



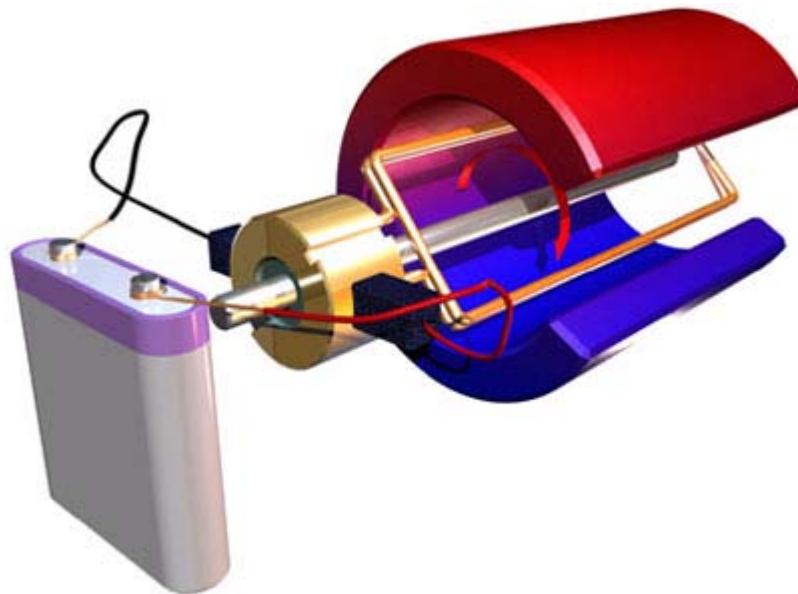
PWM for Speed Control

- If we increase the value in OCR0, the average voltage increases



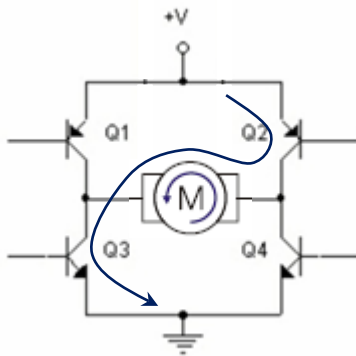
[PWM for Speed Control]

- A change in the duty cycle will change the average voltage and therefore, the speed of the motor

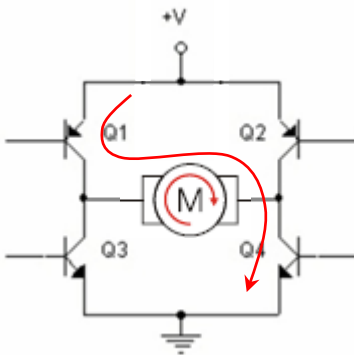


[Speed/Direction Control for the 3π]

■ Direction for motor M1

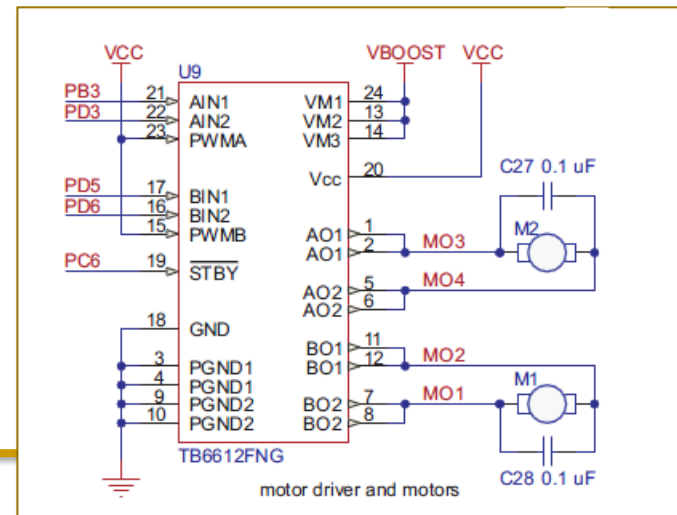


Forward



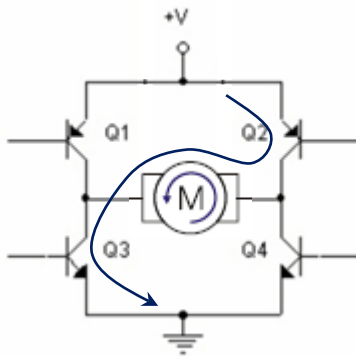
Reverse

PD5	PD6	1	2	3	4	M1
0	0	off	off	off	off	off (coast)
0	1	off	on	on	off	forward
1	0	on	off	off	on	reverse
1	1	off	off	on	on	off (brake)

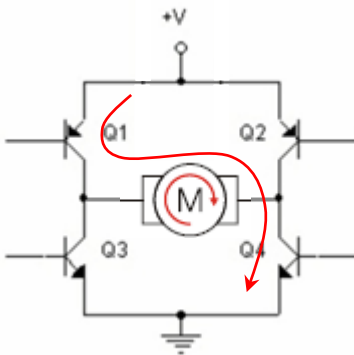


[Speed/Direction Control for the 3π]

■ Direction for motor M2

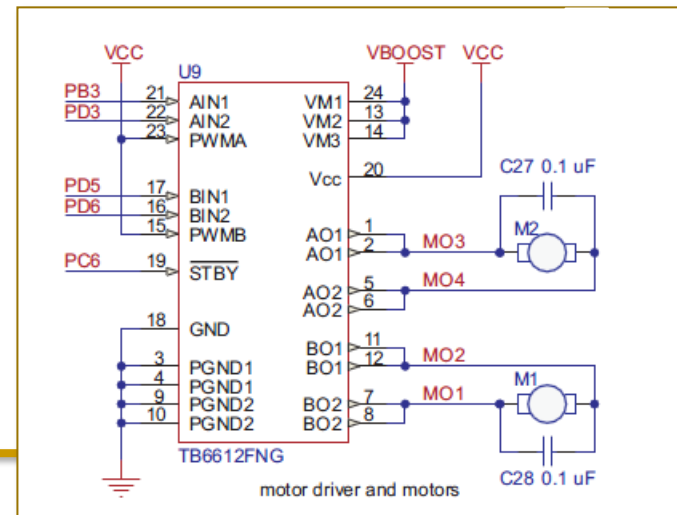


Forward



Reverse

PD3	PB3	1	2	3	4	M2
0	0	off	off	off	off	off (coast)
0	1	off	on	on	off	forward
1	0	on	off	off	on	reverse
1	1	off	off	on	on	off (brake)

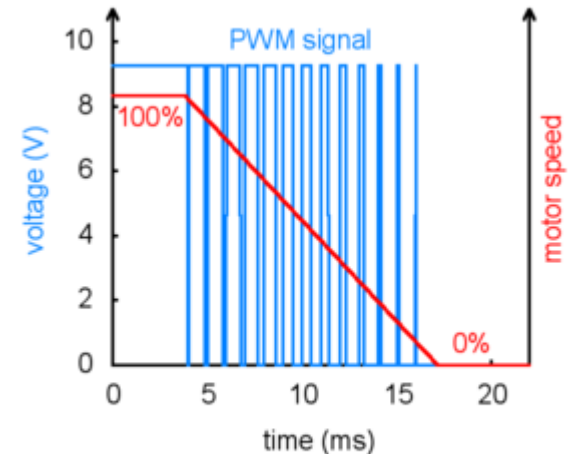


[Speed/Direction Control for the 3π]

■ Speed Control



“Speed control is achieved by rapidly switching the motor between two states in the table. Suppose we keep PD6 high (at 5 V, also called a logical “1”) and have PD5 alternate quickly between low (0 V or “0”) and high. The motor driver will switch between the “forward” and “brake” states, causing M1 to turn forward at a reduced speed. For example, if PD6 is high two thirds of the time (a 67% *duty cycle*), then M1 will turn at approximately 67% of its full speed. Since the motor voltage is a series of pulses of varying width, this method of speed control is called pulse-width modulation (PWM). An example series of PWM pulses is shown in the graph at right: as the size of the pulses decreases from 100% duty cycle down to 0%, the motor speed decreases from full speed down to a stop.”³



PWM speed control, showing gradual deceleration.



[Speed/Direction Control for the 3π]

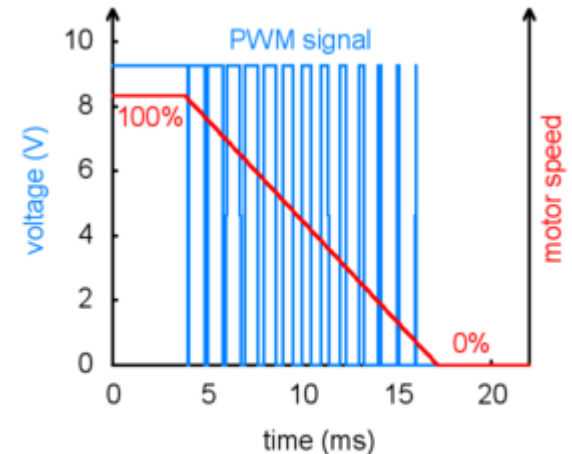
■ Speed Control



“In the 3pi, speed control is accomplished using special PWM outputs of the main microcontroller that are linked to the internal timers Timer0 and Timer2. This means that you can set the PWM duty cycle of the two motors once, and the hardware will continue to produce the PWM signal, in the background, without any further attention.

The `set_motors()` function in the Pololu AVR Library (see [Section 6.a](#) for more information) lets you set the duty cycle, and it uses 8-bit precision: a value of 255 corresponds to 100% duty cycle. For example, to get 67% on M1 and 33% on M2, you would call: ”³

```
set_motors (171,84);
```



PWM speed control, showing gradual deceleration.

Note: 171 = 67% of 255 and 84 = 33% of 255



[Speed/Direction Control for the 3π]

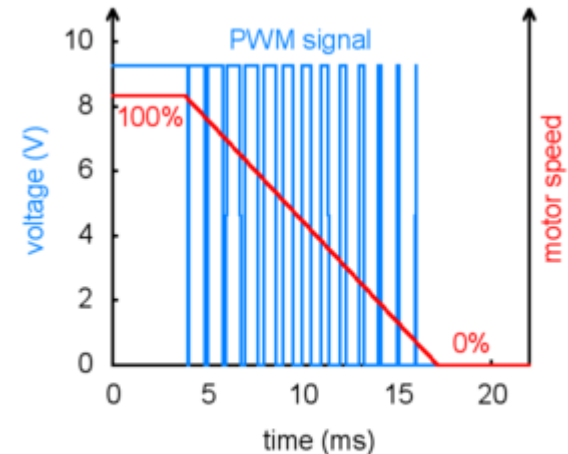
■ Speed Control



“In the 3pi, speed control is accomplished using special PWM outputs of the main microcontroller that are linked to the internal timers Timer0 and Timer2. This means that you can set the PWM duty cycle of the two motors once, and the hardware will continue to produce the PWM signal, in the background, without any further attention.

The `set_motors()` function in the Pololu AVR Library (see [Section 6.a](#) for more information) lets you set the duty cycle, and it uses 8-bit precision: a value of 255 corresponds to 100% duty cycle. For example, to get 67% on M1 and 33% on M2, you would call: ”³

```
set_motors (171,84);
```



PWM speed control, showing gradual deceleration.

Note: To get a slowly decreasing PWM sequence like the one shown in the graph, you would need to write a loop that gradually decreases the motor speed over time.

Note: 171 = 67% of 255 and 84 = 33% of 255





PWM on Timer/Counter 1

(a 16-bit Timer/Counter)



[Phase Correct PWM¹

- We will need the following registers:
 - TCCR1: Timer/Counter Control Register 1
 - TNCT1: Timer/Counter 1
 - ICR1: Input Compare Register 1
 - Sets the duration of the signal (Period or Frequency)
 - OCR1x: Output Compare Register 1
 - Sets the pulse width of the PWM signal
- Can control up to 3 PWM signals with one Timer/Counter



[Phase Correct PWM]

- There are 3 OCR1x registers:
 - OCR1A, OCR1B, OCR1C
- There are 3 OC1x pins on which the generated PWM signals may appear
 - OC1A, OC1B, OC1C

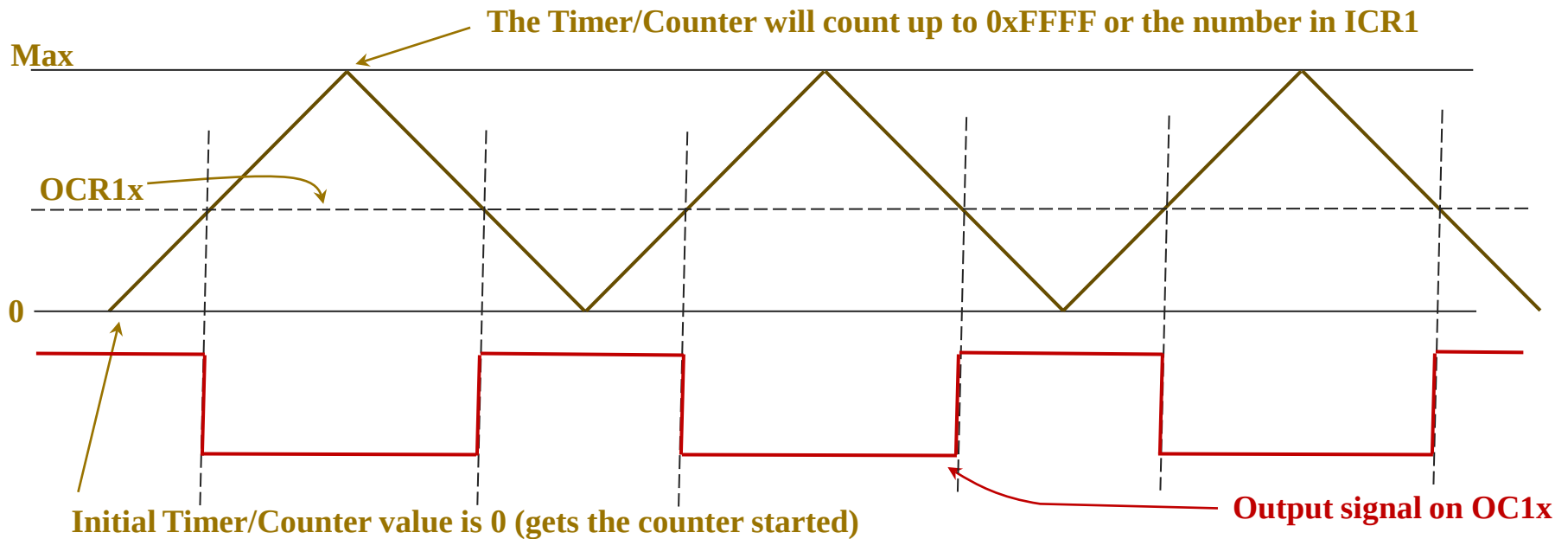


[Phase Correct PWM]

- We will need the following registers:
 - TCCR1: Timer/Counter Control Register 1
 - To initialize the Timer/Counter and set the prescaler
 - TNCT1: Timer/Counter 1
 - To hold the value of the counter
 - ICR1: Input Compare Register 1
 - Holds the upper limit of our counter (makes TC1 more flexible)
 - OCR1x: Output Compare Register 1
 - Holds the count value for which the PWM signal will toggle



[Phase Correct PWM]



Every time the Timer/Counter gets to the value in OCR1x, the output (on OC1x) toggles

There are three OCR1x registers (OCR1A, OCR1B, and OCR1C)

There are three OC1x output lines (OC1A, OC1B, and OC1C)



Calculations

- To calculate the prescaler, the ATmega data sheet gives the following formula (pg129)

$$f_{oc1PWM} = \frac{f_{clk}}{2 * N * TOP}$$

- TOP is the value in ICR1
- For this example, assume a 16MHz system clock and the same servo as before



Calculations

DC Servo Motor Control (DigiSoft: Software for digital world) - Windows Internet Explorer

http://www.digisoft.com.pk/Projects/dc-servo-motor-control

File Edit View Favorites Tools Help

Google position servo motor Go Bookmarks 253 blocked Check AutoLink AutoFill Send to Settings

Favorites Suggested Sites Web Slice Gallery desktop.ini

http://www.coe.uncc.edu/~... DC Servo Motor Control (...)

DigiSoft: Software for digital world

Home

Products

- AVR Project Book
- AVR Project Board
- AVR JTAG
- GPS Simulator and Trainer

DIY Projects

- Text LCD On Project Book
- Graphic LCD Controller
- Swipe Card Reader Interface
- 5 x 7 dotmatrix LED Display
- DC Servo Motor Control**
- Digital RPM Meter for Automobiles
- Incremental Encoder Interface

Technical Notes

Downloads

Sales

Promotions

Contacts

Register

DIY Projects >

DC Servo Motor Control

The R/C Servo Motors are used in model aircraft, R/C Trucks, R/C motorboats and robots.

Radio Control Servo motors are position servo systems that are controlled digitally (through PWM). The usual range is about 180 degrees.

The connections to the motor are

1. GND (Black)
2. VCC (Red)
3. Control (White)

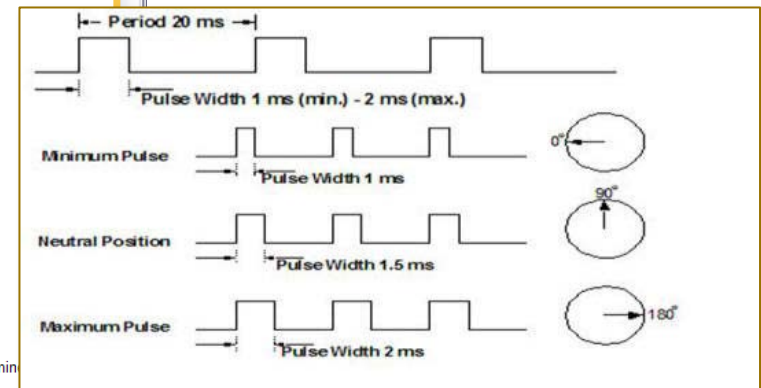
A typical Servo looks like below



The PWM signal to control the position should have a pulse duration between 10-30 msec. and the on time determines position of servo. A 1.5 msec on-time makes the Servo position at the middle (the pulse is given below)

3 pins: Gnd, Vcc, and Control

The PWM signal should have a period of 10 to 30ms. A pulse of 1.5ms sets the neutral position (90°), 1ms sets the left position (0°) and 2ms sets the right position (180°)



Calculations

- For this example¹, assume a 16MHz system clock, and that we want a 20ms period (f=50Hz)
- We need to calculate the value for TOP

$$f_{oc1PWM} = \frac{f_{clk}}{2 * N * TOP}$$

$$TOP = \frac{f_{clk}}{2 * N * f_{oc1PWM}}$$



Calculations

■ For this example TOP could be:

Prescaler N = 1 then TOP(ICR1) = 160000

Prescaler N = 8 then TOP(ICR1) = 20000

Prescaler N = 64 then TOP(ICR1) = 2500

Prescaler N = 256 then TOP(ICR1) = 625

Prescaler N = 1024 then TOP(ICR1) = 156.25



Calculations

- For this example TOP could be:

~~Prescaler N = 1 then TOP(ICR1) = 160000~~

Prescaler N = 8 then TOP(ICR1) = 20000

Prescaler N = 64 then TOP(ICR1) = 2500

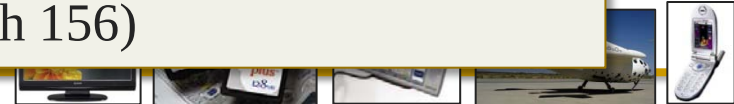
Prescaler N = 256 then TOP(ICR1) = 625

~~Prescaler N = 1024 then TOP(ICR1) = 156.25~~

You cannot use prescaler 1 or 1024 to generate a 50 Hz PWM with a 16 MHz system clock:

Prescaler 1 cannot be used since 160000 too large to fit in TCR1.
(*TCR1 is a 16 bit register with a range from 0 to 65535*)

Prescaler 1024 should not be used since you cannot put decimals into ICR1 (although you could approximate with 156)



[Calculations]

- Chambers¹ suggests:
 - A prescaler of 8
 - Set ICR1 to 20000.
- This will allow you to change OCR1A between 1000 and 2000 to obtain 1 - 2 ms high pulses.
 - Simple, even numbers

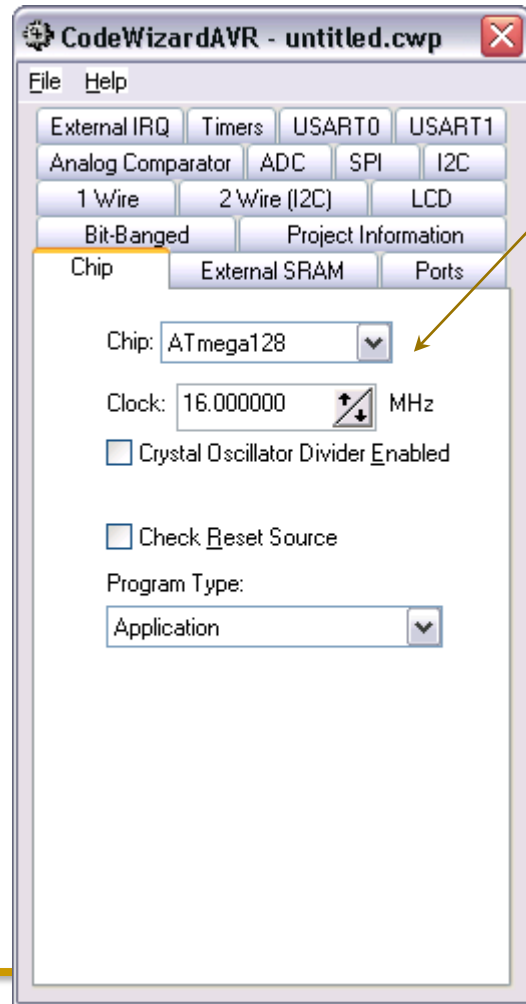


[Setup using CodeVision]

- For this example:
 - ATmega 128
 - 16MHz System Clock
 - 50Hz Timer Clock
 - 1ms (0°)
 - 1.5ms (90°)
 - 2ms (180°)



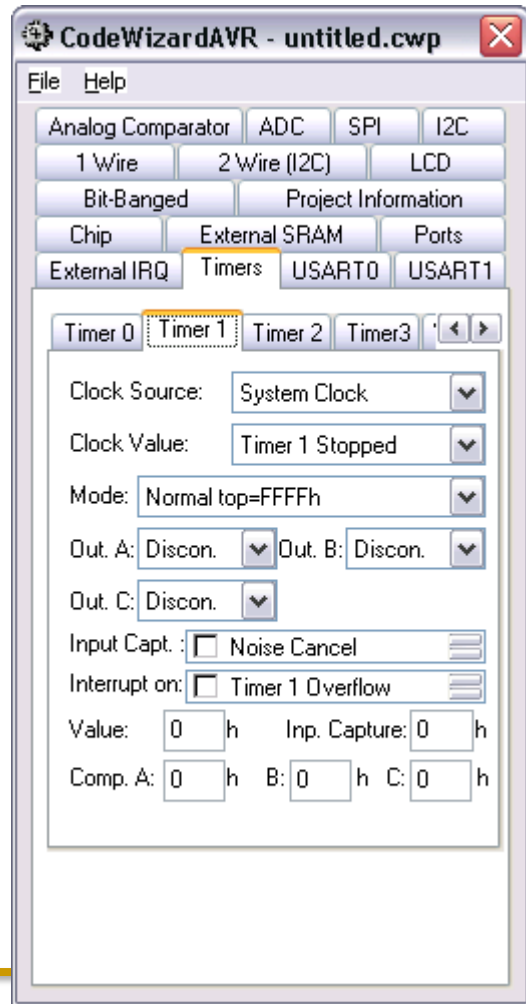
[Setup using CodeVision]



Select the ATmega chip and the frequency



[Setup using CodeVision

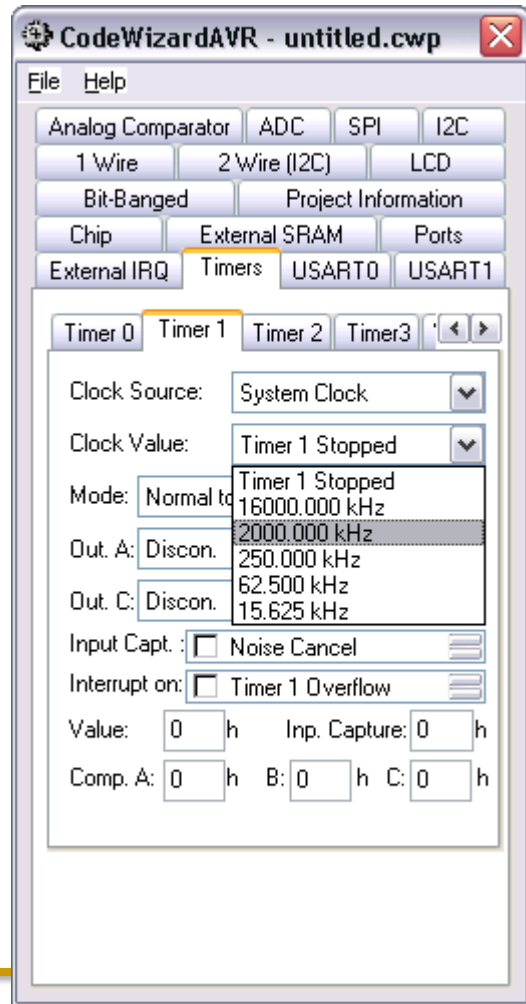


Select the ATmega chip and the frequency

Click on the Timers tab and then on the Timer1 tab



[Setup using CodeVision



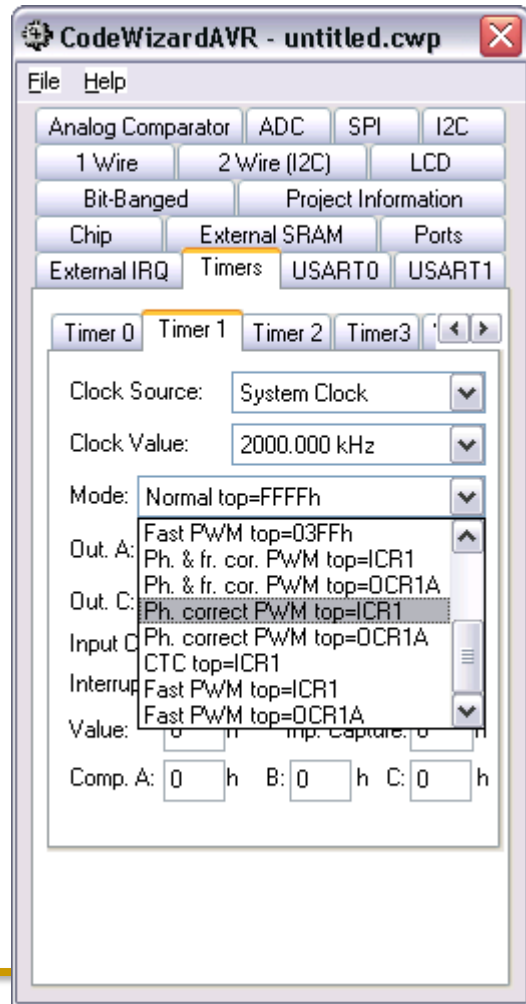
Select 2000.000 kHz for the Clock Value.

These clock values are calculated by taking the System clock you entered on the first page and dividing it by the various prescalers.

What you are actually setting here is what to divide the system clock by in order get these various Timer speeds



[Setup using CodeVision]

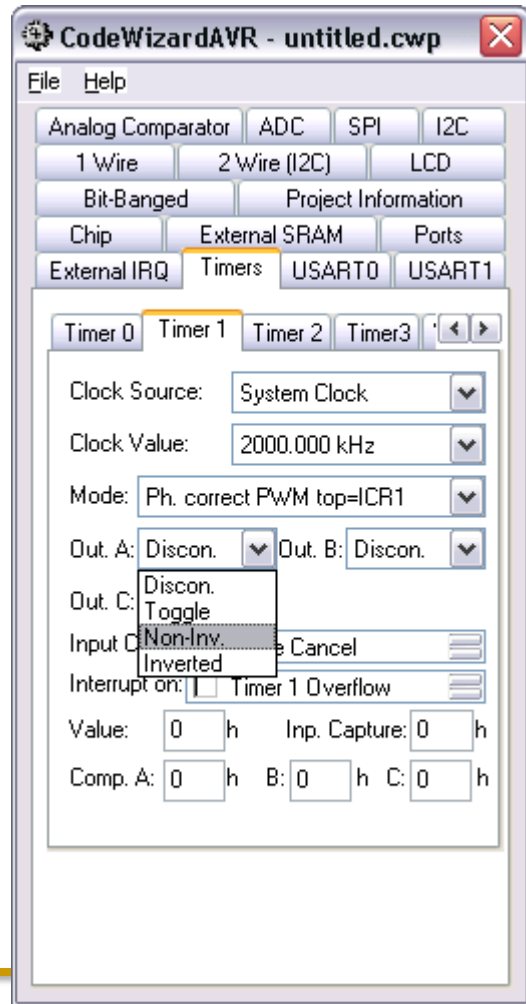


Under Mode, select Ph. & fr. cor. PWM top=ICR1

This is Phase and Frequency correct PWM mode with ICR1 holding the top value.



[Setup using CodeVision]



Timer1 on the ATmega128 can control 3 different servos using Out A, Out B, and Out C.

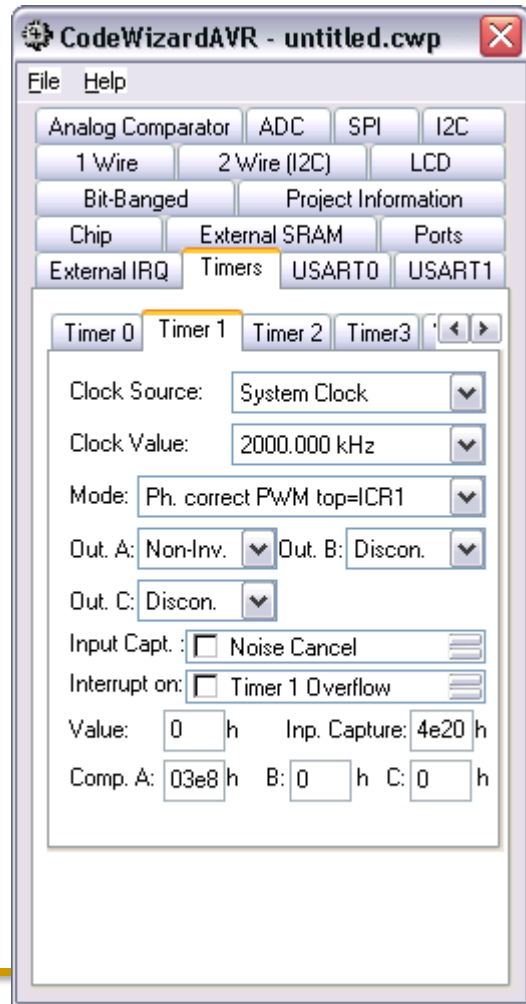
We are only going to control one servo to start off...

Select Non-Inv. from the Out. A pull down menu.

Note: If this option isn't shown, you chose the wrong selection under "Mode"



[Setup using CodeVision



Input Capture is the ICR1 value and allows you to set the TOP value for the timer.

We are going to use ICR1 = 20000 to generate a 50 Hz signal.

CodeWizard requires you to enter the value in HEX (20,000 = 4E20).

(It could be left blank and changed manually in the code)

OCR1A allows you to set the position of the servo.

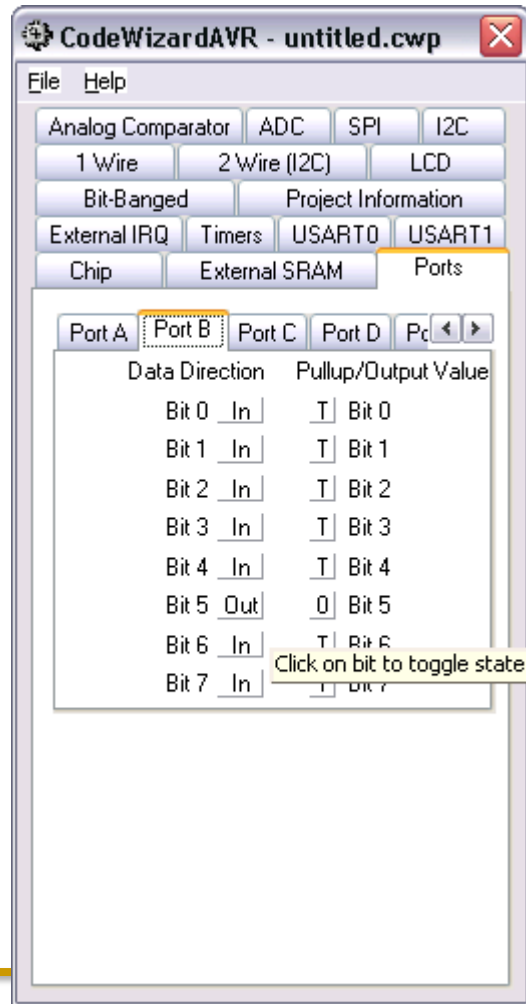
CodeWizard requires you to enter the value in HEX.

(It could be left blank and changed manually in the code)

OCR1A = 1000 (=0x03e8)



[Setup using CodeVision]



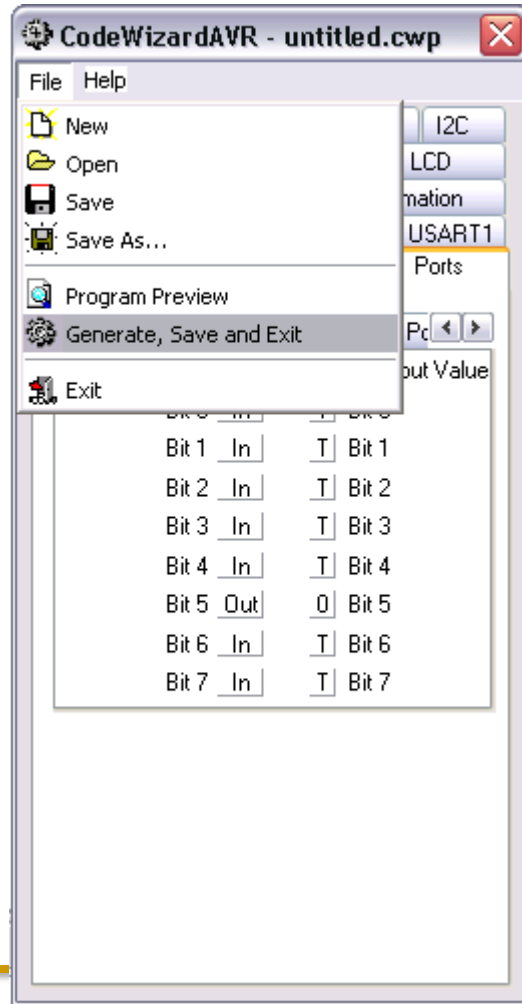
Now we need to set the output pin for our servo. OCR1A is the alternative port function for PORT B.5

Click the Ports tab and then the Port B tab.

Click on the word "In" to change it to "Out"
Leave the output value at 0



[Setup using CodeVision



Click on the File menu and select Generate, Save and Exit to generate the code.

You will be asked the name three files: the .c file, the .prj file, and the .cwj file.



[Setup using CodeVision]

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 2000.000 kHz
// Mode: Ph. correct PWM top=ICR1
// OC1A output: Non-Inv.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0x82;
TCCR1B=0x12;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x4E;
ICR1L=0x20;
OCR1AH=0x03;
OCR1AL=0xE8;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;
```

This is the section of code generated by CodeVision

Note: decimal values could be entered here instead of the hex values in the wizard

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 2000.000 kHz
// Mode: Ph. & fr. cor. PWM top=ICR1
// OC1A output: Non-Inv.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0x80;
TCCR1B=0x12;
TCNT1H=0x00;
TCNT1L=0x00;

ICR1 = 20000;

OCR1A = 1000;

OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;
```



[Setup using CodeVision]

This will position the servo motor:

```
while (1)
{
    OCR1A = 1000;    //position the servo to the left – 0 degrees
    delay_ms (5000); //delay for 5 seconds
    OCR1A = 2000;    //position the servo to the right – 180 degrees
    delay_ms (5000); //delay for 5 seconds
};
```



[Summary]

- In this topic we discussed:
 - The operation of an 8 bit timer/counter in an AVR microcontroller
 - How to use the timer/counter to generate a PWM signal
 - How PWM signals are used to control servo motors
 - Position control
 - Speed control

